

QUESTION PAPER WITH SOLUTION
SUPPLIED FREE WITH BOOK
THE 8051 MICROCONTROLLER BASED
EMBEDDED SYSTEMS

BY

MANISH K PATEL

<http://www.mhhe.com/patel/mbes>

University : VTU
Subject : Microcontrollers
Year of exam : June-July 2013 (VTU)
Semester : 4th SEM B.E. Degree Examination
Subject code : 06EC-42/ 10ES42

JUNE-JULY 2013 VTU
MICROCONTROLLERS
4th SEM B.E. Degree Examination

Part – A

Question 1

a. Compare the CPU architectures:

i. CISC and RISC:

CISC: Complex Instruction Set Computer

In the early days of the computer development, most of the program development work was done in assembly language because high level languages were not developed. Therefore CPU designers tried to make instructions that will perform as much work as possible, which in turn has led to development of instructions with many addressing modes for arithmetic, logical and data transfer operations. These powerful instructions provide greater flexibility in performing operations. The CISC architecture require less the number of instructions per program at the cost of number of cycles per instruction. The 8051 microcontroller is based on CISC architecture, for example multiply and divide instructions are complex instructions for which operation is performed in hardware.

The common characteristics of CISC architecture are:

- Complex hardware: complex as well as more addressing modes, variable instruction size.
- Many clock cycles to execute an instruction.
- High code density- small program size.
- Complex data types.

RISC: Reduced Instruction Set Computer

The simple instructions which perform a few operations at a time will provide high performance because of less hardware requirements for instruction decoder. Thus, instructions require very

The 8051 Microcontroller based Embedded Systems, First Edition.
<http://www.mhhe.com/patel/mbes>

less time to execute. The RISC instructions have few addressing modes for supported by all instructions. It reduces the cycles per instruction at the cost of the number of instructions per program. The Microchip PIC microcontrollers are based on RISC architecture.

The common characteristics of RISC architecture are:

- Simple hardware: simple and less addressing modes, fix instruction size.
- Single clock cycle execution, uniform instruction format.
- Low code density- Larger program size.
- Few data types in hardware.
- Emphasis is on software: Compiler design is more complex.

ii. von-Neumann and Harvard

Von Neumann Architecture: It has single memory storage to hold both program instructions and data i.e. common program and data space. The CPU can either read an instruction or data from the memory one at a time (or write data to memory) because instructions and data are accessed using same bus system. The Von Neumann Architecture is named after the mathematician and computer scientist John Von Neumann. The basic organization of memory in this architecture is shown in figure 1.7.

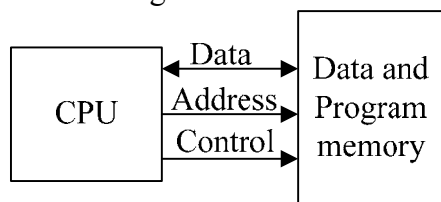


Figure 1.7 Von Neumann architecture

The advantage of Von Neumann architecture is simple design of microcontroller chip because only one memory is to be implemented which in turn reduces required hardware. The disadvantage is slower execution of a program. It is also referred as Princeton architecture as it was developed at Princeton University. Motorola 68HC11 microcontroller is based on Von Neumann architecture.

Harvard Architecture: It has physically separate memory storage to hold program instructions and data i.e. separate program and data space. Since it has separate buses to access program and data memory, it is possible to access program memory and data memory simultaneously. The organization of memory and buses in this architecture is shown in figure 1.8.

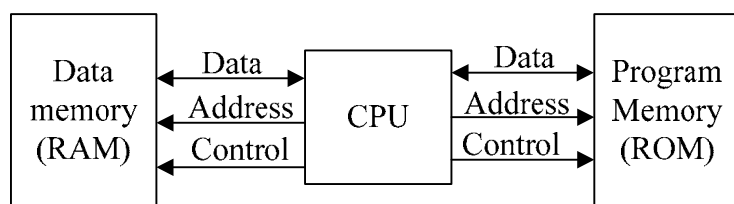


Figure 1.8 Harvard architecture

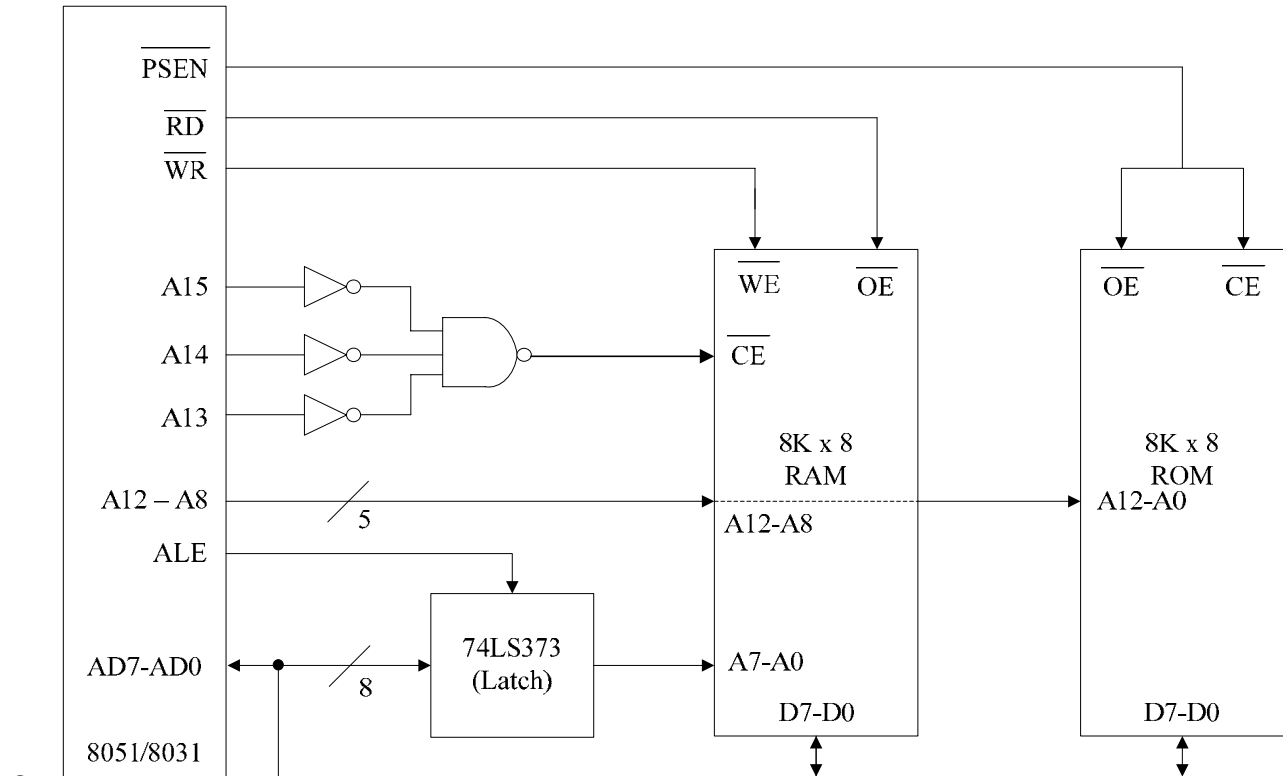
The advantage of a Harvard architecture microcontroller is that it is faster for a given circuit complexity because it offers greater amount of parallelism. The disadvantage is that it requires more hardware, because two sets of buses and memory blocks are required. MCS 51 (8051 family) and PIC microcontrollers are based on Harvard architecture.

iii. Microprocessor with Microcontroller

Microprocessor	Microcontroller
Microprocessor is complete functional CPU i.e. it contains ALU, registers, stack pointer, program counter, instruction decode and control unit and interrupt processing circuits.	Microcontroller is complete functional microcomputer i.e. it contains the circuitry of microprocessor and in addition it has built in memory (ROM, RAM), I/O circuits and peripherals necessary for an application.
Microprocessor instruction sets are data processing intensive, means powerful addressing modes and many instructions to move data between memory and CPU to handle large volumes of data.	Microcontrollers have instruction sets that are related to the control of inputs and outputs, means they have many bit handling instructions along with byte processing instructions.
Microprocessor based products are primarily designed to interact with humans and are more flexible to design	Microcontroller based products are primarily designed to interact with machines; once a system is designed they are less flexible.
Access times for external memory and I/O devices are more, resulting in a slower system.	Access times for on-chip memory and I/O devices are less, resulting in a faster system.
Microprocessor based systems require support devices and are usually bulkier, costly, less reliable and consume more power.	Microcontroller based systems require less external hardware, reducing PCB size and hence are compact, cheaper, more reliable and consume less power.
Software protection is not possible because of the requirement of external code memory.	Software protection is possible because of on-chip code memory.

b. Interface 8051 to external ROM and RAM and explain how 8051 access them?

Interfacing of 8Kbytes of SRAM (6264) and 8Kbytes of EEPROM (2864) with the 8051 both starting at address 0000H is shown below.



Interfacing RAM as well as ROM with 8051

The 8051 has two parallel address spaces for RAM and ROM ranging from 0000H to FFFFH, therefore we can connect both RAM and ROM at same address range. Different control signals are used to access data from both memories.

The 8KByte chip requires 13 address lines ($A_{12}-A_0$). RAM signals \overline{WE} and \overline{OE} are connected with \overline{WR} and \overline{RD} of the 8051 respectively and ROM signals \overline{OE} and \overline{CE} are connected with \overline{PSEN} of the 8051 as shown in figure.

The 8051 has 64 Kbyte data memory space. It is accessed using 'MOVX' instructions. \overline{RD} and \overline{WR} signals from the 8051 are used to access RAM.

$\overline{\text{PSEN}}$ is generated either when program memory is accessed by 'MOVC' instructions or during program byte fetches. It may also be connected with $\overline{\text{CE}}$ (chip enable) of memory chip if only one ROM chip is present in a system.

Refer example 21.12 for detailed description.

Question 2

a. Explain briefly the five addressing modes of 8051 with example for each.

Addressing modes
1. Immediate addressing
2. Register addressing
3. Direct addressing
4. Indirect addressing <ul style="list-style-type: none">• Register indirect addressing• Indexed addressing

1. Immediate addressing mode

The data (constant) is specified as a part of instruction in a program memory. The data is available immediately as a part of instruction itself, therefore immediate addressing is very fast. However, since the data is fixed, at run time it is not flexible. The instructions using an immediate operand have an 8 bit or 16 bit number following the op-code. For example,

```
MOV A, #data      // load 8 bit immediate data in to Accumulator
MOV A, #55H       // A= 55H
```

```
MOV Rn, #data     // load 8 bit immediate data into Rn
MOV R3, #0FFH     // R3= FFH
```

```
MOV DPTR, #data16 // load 16-bit number into DPTR
MOV DPTR, #2000H  // DPTR=2000H
```

2. Register addressing mode

In register addressing mode, the operands are specified by register names. Register A and R0 to R7 may be named as a part of the instruction mnemonic. The advantage of register addressing mode is that it occupies only one byte memory, and is fast because only on-chip registers are accessed i.e. instruction takes only one machine cycle for execution. For example,

```
MOV A, Rn // copy contents of register Rn to Accumulator
MOV A, R2 // If R2=10H → A=10H
```

```
MOV Rn, A // copy contents of Accumulator to register Rn
MOV R1, A // If A=20H → R1=20H
```

3. Direct addressing mode

The data is accessed directly from the memory address specified as one of the operand i.e. one of the operand is an 8-bit address for internal RAM location. Internal RAM includes 128 byte of RAM from (00H-7FH) and any special function register. It is more flexible compared to immediate and register addressing because the value to be accessed from address may be variable. These are 2 byte instructions (3 bytes when source and destination are both direct addresses). The address refers to either byte location or a specific bit in a bit addressable byte. For example,

```
MOV A, direct // copy data from (contents of) address direct in to A
MOV A, 10H // If address 10H contains data 50H i.e (10H)=50H → A=50H
```

```
MOV direct, A // copy data from A to address direct
MOV 10H, A // If A=44H → (10H)= 44H
```

```
MOV Rn, direct // copy data from address direct in to register Rn
MOV R5, 80H // If (80H)= FFH → R5= FFH
```

```
MOV direct, Rn // copy data from Rn to address direct
MOV 50H, R3 // R3=10H → (50H)= 10H
```

```
MOV direct, #data // load 8 bit immediate data in to address direct
MOV 0A0H, #20H // (A0H) = 20H
```

```
MOV direct1, direct2 // copy data from address direct2 to address direct1
MOV 50H, 83H // If (83H)=10H → (50H)=10H
```

4. Indirect addressing mode

The data is specified indirectly in an instruction i.e. address of the data (rather than data itself) is specified as one of the operand. Here, the register holds the address that contains the data i.e. the number in the register is treated as an pointer to address. Indirect addressing mode is the most flexible and useful in array operations. There are two types of indirect addressing in the 8051

Register indirect addressing mode: The register indirect addressing uses only register R0 or R1 to hold address of the data in internal RAM, these two registers are also referred to as pointer registers or simply pointers. The symbol @ is used along with R0 or R1 to indicate indirect addressing. For example,

```
MOV @Ri, #data    // load constant value in to address contained in Ri
MOV @R0, #30H      // If R0=40H, → (40H)=30H
```

```
MOV @Ri, direct    // copy data form address direct to address Ri
MOV @R1, 10H        // If (10H)=50H, R1=15H→ (15H)=50H
```

```
MOV direct, @Ri     // copy data from address in Ri to address direct
MOV 10H, @R1        // If R1=50H, (50H)=15H→ (10H)=15H
```

```
MOV @Ri, A          // copy data from A to address in Ri
MOV @R0, A           // If A=50H, R0=15H→ (15H)=50H
```

```
MOV A, @Ri          // copy data from address in Ri to A
MOV A, @R1           // If R1=50H, (50H)=15H→ A=15H
```

Indexed addressing mode: Two registers are used to form the address of the data. The contents of either DPTR or PC are used as a base address and the A is used as index (or offset) address. The final address is formed by adding these two registers. It results in a forward reference of 0 to 255 bytes from the base address. They are used to access only program memory (internal as well as external)

Indexed addressing is used to access data tables (lookup tables) from the program memory and implementing jump tables. They are also suitable for multidimensional array operations.

The instructions are:

```
MOVC A, @A+PC      // copy data (or code) byte from program memory address formed by
                   // addition of contents of A and PC into A
```

```
MOVC A, @A+DPTR    // copy data (or code) byte from program memory address formed by
                   // addition of contents of A and DPTR into A
```

b. After reset, the contents of internal memory of 8051 with address 0AH and 0BH contains data 22H and 33H, respectively. Sketch the contents of internal memory from address 07H to 0BH and the value of register SP, after executing the following code:

PUSH 0AH

:

MOV 81H, #0BH

POP 09H.

SP	0AH
07H	00H
08H	22H
09H	33H
0AH	22H
0BH	33H

- c. Write a subroutine which checks the content of 20H. If it is a positive number, the subroutine find its two's complement and store it in same location and returns.**

```
MOV A, 20H           // transfer content of 20H to A
RLC A                // move MSB to carry
JC EXIT              // checking carry for +ve or -ve number
CPL A                // if +ve then one's complement
ADD A, #01H          // two's complement
MOV 20H, A           // converted value transfer to the same location
EXIT:NOP
HERE:SJMP HERE
```

Question 3

- a. What are assembler directives? Explain any four of them.**

The assembler directives are the statements that directs the assembler what to do during assembling. They reserve memory space for data, define constants, and tell assembler where to assemble program in a memory. They are also referred as pseudo instructions statements as they are effective only during the assembly of the program but they do not generate any machine code.

Originate- ORG

The ORG directive allows us to place the code and data anywhere in the program address space. The number after the ORG can be in hex or decimal. If the number is decimal, the assembler will

The 8051 Microcontroller based Embedded Systems, First Edition.
<http://www.mhhe.com/patel/mbes>

convert it to hex. Its format is ‘ORG Address’ For example, ORG 0000H will place the instructions (or data) from address 0000H onwards (some assembler use this directive as ‘.ORG Address’)

Define Byte- DB

This directive defines the byte type variable. When DB is used to define data, the numbers can be in decimal, binary, hex, or ASCII formats. For the binary numbers B is as a suffix. Similarly, H is used after hexadecimal numbers. Irrespective of the type of the byte, the assembler will convert the byte to hex. To indicate ASCII numbers, the characters are placed in quotation marks (‘character’). The DB is also used to allocate memory in byte sized chunks.

Define Word- DW

This directive defines 16 bit variable. (The meaning of “Word” is microcontroller/ processor dependent, for example, for 32 bit controller, word means 32 bits)

Equate- EQU

Equates label to the number.

EQU TEMP, #20 will assign name TEMP to data #20.

END

End directive tells the assembler to stop assembling.

There are still many other directives like DD, DBIT, DS, PROC, ENDP, LABEL, USING, EXRERN, IF, ELSE, ELSEIF, ENDF, NAME, PUBLIC, SEGMENT, BSEG, CSEG, BIT, CODE, DATA, EVEN etc. used with different assemblers. Programmer should refer documents or help for the assembler in use.

b. If the XTAL frequency of 8051 is 8 MHz, find the time taken to execute the following program:

```
MOV R2,#04
MOV R1,#06
WAIT:  DJNZ R2, WAIT.
```

	MOV R2, #04	Machine cycle-1
	MOV R1, #06	Machine cycle-1
WAIT:	DJNZ R2, WAIT	Machine cycle-2×4

Total cycle: $10 \times 12 \text{ clock cycle} / 8 \text{ MHz} = 15\mu\text{Sec}$

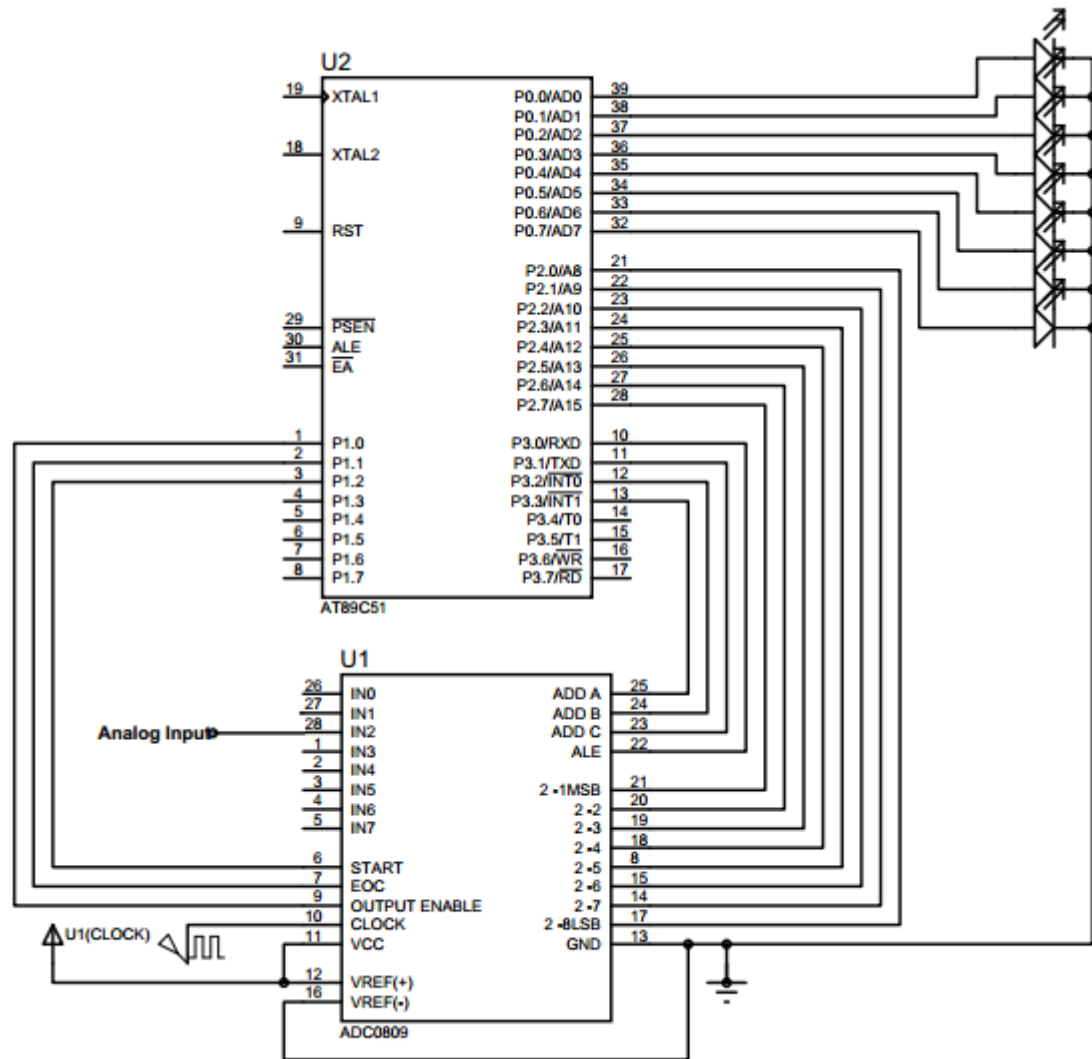
- c. Write 8051 APL which checks whether the ten numbers stored from external RAM memory address, 2000H are odd/even. The program should store accordingly 00H/FFH from internal location 30H onwards.**

```

                MOV R0,#30H           //Starting address of internal memory
                MOV R7,#10            //total numbers
                MOV DPTR,#2000H       //starting address of external memory
AGAIN:          MOVX A,@DPTR          //transfer number from external memory
                RRC A                 //transfer LSB to Carry bit
                JC ODD_NO             //Jump if carry is set means number is odd
                MOV @R0,#0FFH         //Transfer FFH if Even
                SJMP NEXT
ODD_NO:         MOV @R0,#00H          //Transfer 00H if number is even
NEXT:          INC R1                 //Increase internal memory index
                INC DPTR              //Increase external memory index
                DJNZ R7,AGAIN          //Repeat for 10 times
```

Question 4

- a. Interface ADC0809 to 8051 and write ALP to convert the analog voltage connected to second channel. Display the digital value on LEDs connected to Port-0.**



// START OF CONVERSION=P1.2, EOC=P1.1, OE=P1.0, ALE=P3.0

// ADD A=P3.3, ADD B=P3.2, ADD C=P3.1

// LEDs ON P0, CONVERTED DATA FROM ADC0809 = P2, ANALOG INPUT = IN2 CHANNEL 2

```
MOV P2,#0FFH    // P2 as input port
MOV P0,#00H     // switch off LEDs connected to port 1
MOV P3,#04H     //
```

MAIN:

```
CLR P3.1        //SELECT CHANNEL 2 BY ADD A,B,C
```

```

SETB P3.2
CLR P3.3

AGAIN:   CLR P1.0 ;OE
         CLR P3.0;ALE
         SETB P3.0 ;ALE
         SETB P1.2 ;START
         CLR P1.2;START
HERE1:   JB P1.1,HERE1    //wait for end of conversion
         SETB P1.0        // OE
         MOV A,P2
         MOV P0,A
         ACALL DELAY
         SJMP AGAIN
DELAY:   NOP
UB2:     MOV R6, #3FH
UB1:     MOV R5, #5FH
HERE:    DJNZ R5, HERE
         DJNZ R6, UB1
         DJNZ R7, UB2
         RET

```

b. Interface 8051 to stepper motor and write an ALP to rotate the motor first +4 steps and then -6 steps.

```

ORG 0000H
MOV A, #0CCH    // full step sequence, the first entry in table is 1011= C.
                // Here CC is used for continuous rotation.

MOV R0,#04
REPEAT: MOV P2, A    // send sequence to motor
        RR A        // next step of sequence to rotate in clock wise rotation.
        LCALL DELAY // wait before going to next step
        DJNZ R0, REPEAT // next step

MOV R0,#06
REPEAT: MOV P2, A    // send sequence to motor
        RL A        // next step of sequence to rotate in ANTI-clock
                //wise rotation.
        LCALL DELAY // wait before going to next step
        DJNZ R0, REPEAT // next step

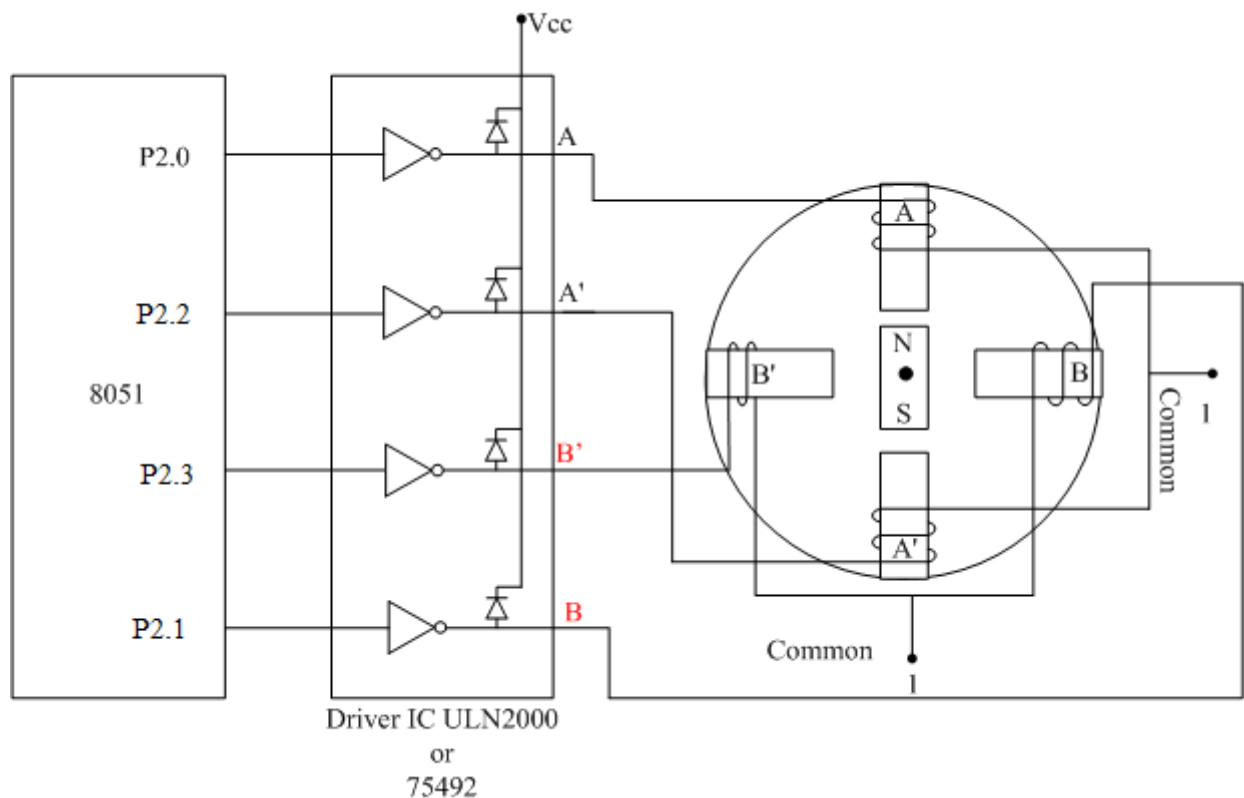
```

The 8051 Microcontroller based Embedded Systems, First Edition.
<http://www.mhhe.com/patel/mbes>

```

DELAY:    MOV R1, # 60H    // delay, it can be varied to change speed of rotation
THERE:    MOV R2, #0FFH
HERE:     DJNZ R2, HERE
          DJNZ R1, THERE
          RET
          END

```



PART – B

Question 5

- a. What is the difference between timer and counter operation of 8051? How to start/stop the timer/counter of 8051 when
- GATE control is not used
 - GATE control is used

The major difference between interval timer and event counter is source of the clock pulse, when timers are used as a timer (interval timer), the internal clock signal is used as a clock source for the interval timer activities,

When timers are used as event counters, pin T0/T1 (P3.4/P3.5) are basically used to provide external pulses for timer 0/1, therefore external pulses will increment the timer registers TLX and THX. The timer can be configured as an event counter by setting $C/\bar{T} = 1$ in TMOD register. The other difference between interval timer and event counter is that counter is normally started with initial value of “0000” so TLX and THX normally initialized with value 00H.

The selection of the clock source using C/\bar{T} bit is illustrated in the following figure. For timer clock to reach timer stages the C/\bar{T} bit must be 0 (interval timer operation), bit TRX (TR0 or TR1) must be set to 1 (timer run) and gate bit in TMOD register must be 0 or \overline{INTX} pin must be 1. We may conclude that when timer is configured as an interval timer, the timer clock pulses are fed to timer stages when $C/\bar{T} = 0$, timer run bit TRX=1 AND Gate =0 or external input pins $\overline{INTX}=1$. The path for clock signal when $C/\bar{T} = 0$ is shown by dotted line figure 14.4 (a).

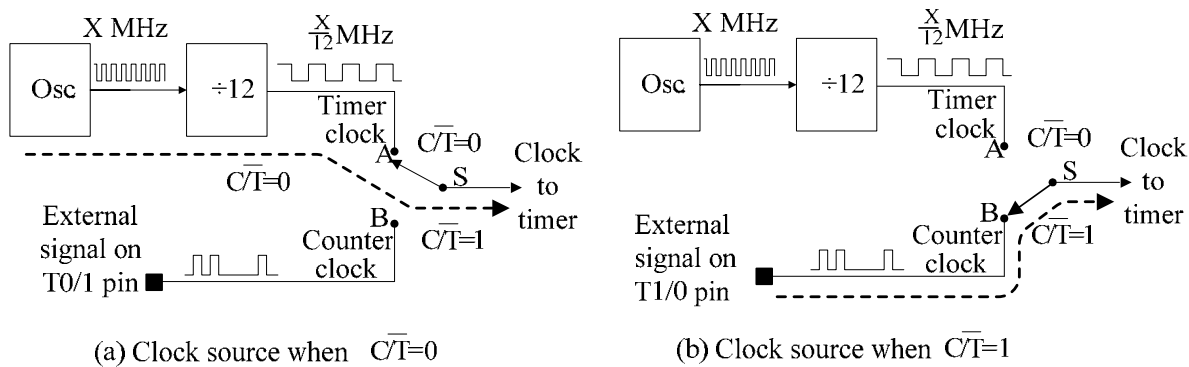


Figure 14.4 Clock source selection using C/\bar{T} bit

When $C/\bar{T} = 1$, selection switch S will be connected to point B. The point B is connected to timer input pins T0/1. The clock source for the timer circuits is external pulses on pin T0 (Pin 14) or T1 (Pin 15) of microcontroller for timer 0 and timer 1 respectively. The path for clock signal when $C/\bar{T} = 1$ is shown by dotted line figure 14.4 (b). The other control bits for timer start/stop will remain same as described in the above section.

i) GATE control is not used : When GATE control is not used, it is referred as software control of the timer/Counter

The software control of the timers is illustrated in the following figure (a). The software control (start/stop of timers) is achieved when Gate bit is programmed to zero i.e. Gate=0. When

Gate=0, the output of NOT gate is 1, which is applied to OR gate, since one of the input of OR gate is 1, its output will be 1 irrespective of status of the other inputs. The output of OR gate (=1) is given to AND gate. Now one input of AND gate is 1, therefore output of AND gate is controlled by other input. The other input is TR0/1 bit, therefore output of AND gate is controlled by status of TR0/1 bit i.e. If TR0/1=0, output of AND gate is 0 and If TR0/1=1, output of AND gate is 1. The output of AND gate is used to control the switch which will connect clock signal to the timer stages for their operation. When output of AND gate is 1, the switch is closed and the clock signal is applied to the timer stage and the timer starts its operation. Therefore, effectively we can say that TR0/1 bits only controls the timer operation. See dotted line in the figure 14.2(a). As mentioned earlier, TR0/1 bit can be set or cleared using SETB (SETB TR0 or SETB TR1) and CLR (CLR TR0 or CLR TR1) instructions.

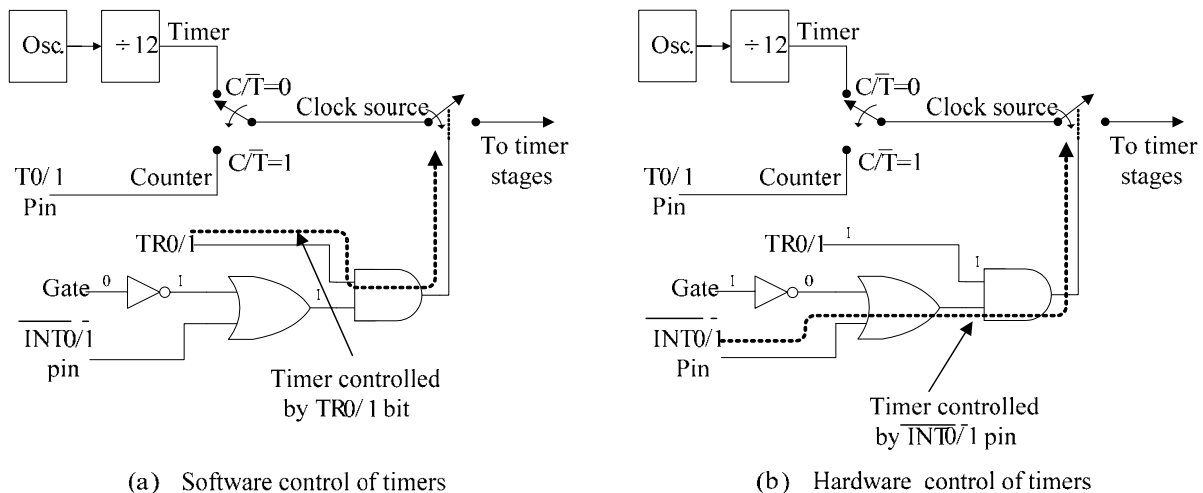


Figure Software/hardware control of timers

ii) GATE control is used : When GATE control is used, it is referred as hardware control of the timer/Counter

The hardware control of the timers is illustrated in figure 14.2 (b). The hardware control of timers is achieved when Gate bit is programmed to one i.e. Gate=1 and TR0/1=1. When Gate=1, the output of NOT gate is 0, which is applied to OR gate, since one of the input of OR gate is 0, the output is controlled by the second input. The second input to OR gate is given from $\overline{INT0}$ and $\overline{INT1}$ pins for timer 0 and timer 1 respectively. When $\overline{INT0/1} = 0$, the output of OR gate is 0 and when $\overline{INT0/1} = 1$, the output of OR gate is 1. Since TR0/1 is already programmed to be 1, the status of $\overline{INT0/1}$ pin will determine the output of AND gate and finally operation of the timer because output of AND gate is used to connect the clock source to the timers. See dotted line in figure 14.2(b)

This feature allows the pulse width measurement i.e. a pulse applied to $\overline{\text{INTX}}$ pin will run the timer as long as it is high, then, the timer register may be read to determine time period of the pulse. It should be noted that the external hardware control is achieved by making corresponding timer runs bits TR 0/1=1 as shown in figure 14.2 (b).

b. Explain briefly the interrupts of 8051, indicate their vector addresses.

Five interrupts are available in the 8051. Three of them are internal interrupts i.e. they are generated because of internal operation of the 8051. They are timer 0 (TF0), timer1 (TF1) and serial port (TI or RI) interrupts. Remaining two are external interrupts $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ i.e. they are invoked by external signals given to pins $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$. The bar over INT0 and INT1 indicate that they are active low interrupt inputs. The external interrupts $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ are also referred as IE0 and IE1 respectively. For each interrupt source there is a fixed location in a program memory that contains its interrupt service routine (ISR). This part of memory which stores the ISRs is called the interrupt vector table. It is shown in Table 16.1.

Table 16.1 Interrupt vector table of the 8051

Interrupt source	Interrupt vector	Interrupt type	Interrupt flag clearing
External interrupt 0 ($\overline{\text{INT0}}$)	0003H	External	Auto*
Timer 0 interrupt (TF0)	000BH	Internal	Auto
External interrupt 1 ($\overline{\text{INT1}}$)	0013H	External	Auto*
Timer 1 interrupt (TF1)	001BH	Internal	Auto
Serial port interrupt (TI or RI)	0023H	Internal	By program

* For edge triggered external interrupts only.

When any interrupt is generated (or asserted), it forces the microcontroller to jump to fixed address in vector table. For example when INT0 is asserted, microcontroller will automatically jump to memory address 0003H, similarly for timer1 interrupt, it will jump to address 001BH in interrupt vector table. Since program execution is transferred to fixed location corresponding to each interrupt source, the interrupts in the 8051 are also referred as vectored interrupts. The interrupt vector table of the 8051 is shown in figure 16.1.

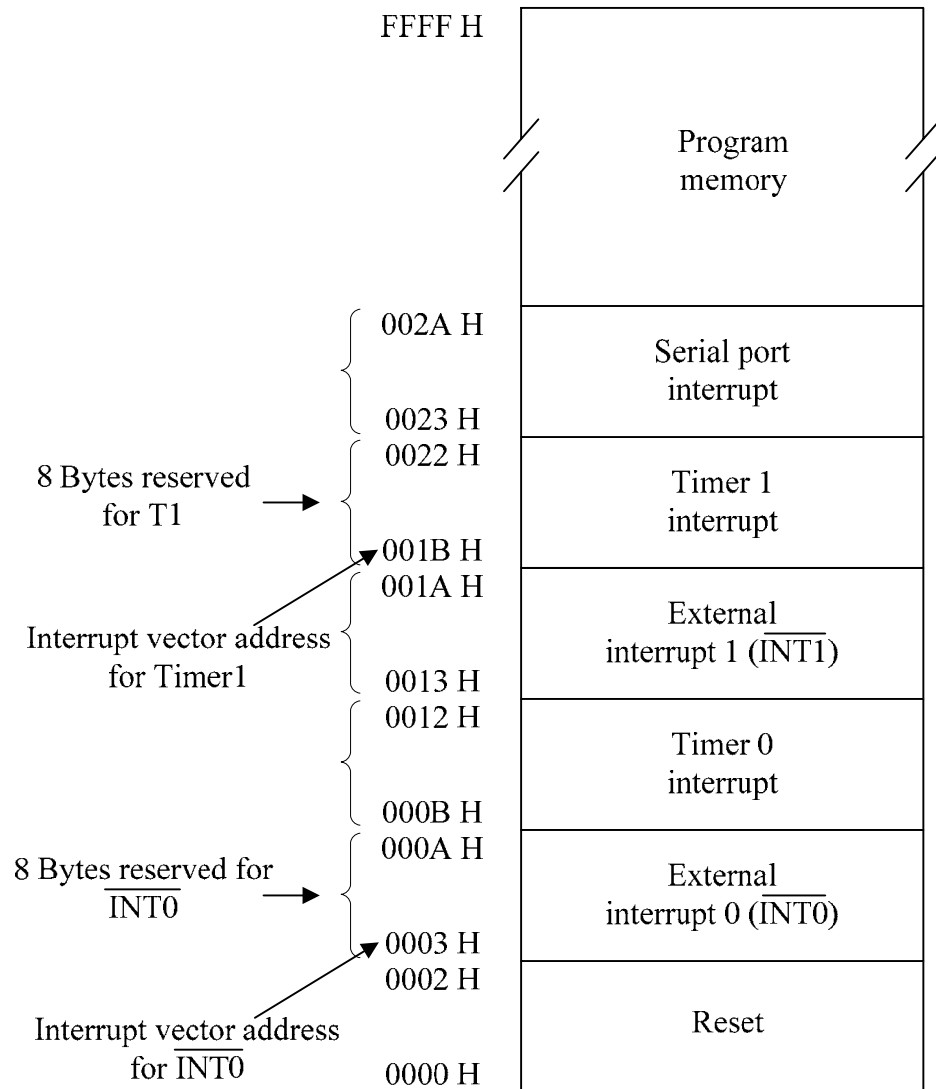


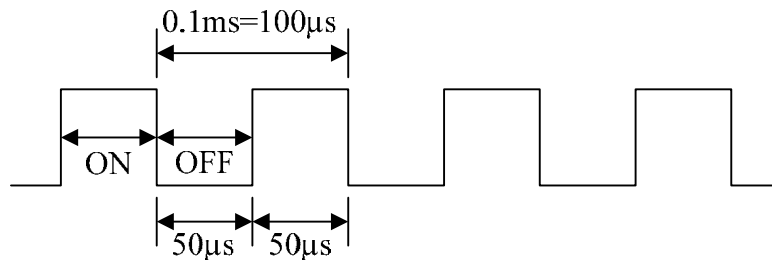
Figure 16.1 Interrupt vector table of the 8051

It can be observed from figure 16.1 that there are only eight bytes reserved for ISR of each interrupt. If ISR requires 8 or less bytes it can be directly written in interrupt vector table, otherwise LJMP (or AJMP) instruction is written at corresponding address in the vector table to jump to actual address of the ISR. For example, consider ISR for $\overline{\text{INT0}}$ is larger than 8 bytes, and ISR is written at address 1000H. So we have to write LJMP instruction at memory location 0003H, which in turn will jump to location 1000H. When $\overline{\text{INT0}}$ interrupt is asserted, it will force the program execution to start at address 0003H, at this address LJMP instruction is written which in turn divert the program execution at address 1000H (see figure 16.2) and ISR is executed. This approach will increase the interrupt response time.

For detail Discussion refer topic 16.2

- c. Write an ALP in 8051 which generates a square wave of frequency 10 kHz on pin P1.2, using timer-1. Assume XTAL frequency as 11.0592 MHz. What is the minimum frequency that can be generated?**

The period of square wave is $1/10 \text{ KHz} = 0.1 \text{ ms}$.



Square wave have duty cycle of 50%, so, ON period = OFF period = Total Period/2
 $= 100 \mu\text{s} / 2 = 50 \mu\text{s}$

We should generate the time delay of $50 \mu\text{s}$ and toggle pin P2.0 every $50 \mu\text{s}$ to generate square wave of 1 KHz.

The frequency of timer clock = $11.0592 \text{ MHz} / 12 = 0.9215 \text{ MHz}$

\therefore The time period of timer clock = $1 / 0.9215 \text{ MHz} = 1.085 \mu\text{s}$ (time period of one machine cycle)
also, Timer register will be incremented by one every $1.085 \mu\text{s}$

Number of timer clock pulses (of $1.085 \mu\text{s}$) required to make $50 \mu\text{s} = 50 / 1.085 = 46.08 = 46$ (approx)

If timer is used in mode 1

The Count to be loaded in to TH and TL = $65536 - 46$ (or $65535 - 46 + 1$) = $65490 = \text{FFD2H}$
 $\therefore \text{TH1} = \text{0FFH}$ and $\text{TL1} = \text{D2H}$.

The steps to develop the program to generate the square wave are as follows:

- Configure any one timer as an interval timer using TMOD register
- Load initial count in to timer registers THX-TLX to get desired delay
- Start the timer
- Wait until timer overflows

- Toggle port pin on which square wave should be generated
- Stop timer and clear the overflow flag and repeat the process continuously

The program to generate the required square wave is given below,

```

                MOV TMOD, #10H           // configure Timer 1 as interval timer and in
                                           // mode 1
REPEAT:        MOV TL1, #0D2H           // load count in timer registers TH1-TL1
                MOV TH1, #0FFH
                SETB TR1                 // start timer 1
WAIT:          JNB TF1, WAIT             // wait until timer overflows (wait for 50us)
                CLR TR1                 // stop timer
                CLR TF1                 // clear timer overflow flag
                CPL P1.2                 // toggle P1.2 to get square wave
                SJMP REPEAT              // repeat above steps to generate square wave

```

If timer is used in mode 2

The Count to be loaded in to TH = $256 - 46$ (or $255 - 46 + 1$) = 65490 = D2H
 \therefore TH1=D2H

The program to generate the required square wave is given below,

```

                MOV TMOD, #20H           // configure Timer 1 as interval timer and in
                                           // mode 1
REPEAT:        MOV TH1, #0D2H           // load count in timer registers TH1
                SETB TR1                 // start timer 1
WAIT:          JNB TF1, WAIT             // wait until timer overflows (wait for 50us)
                CLR TF1                 // clear timer overflow flag
                CPL P1.2                 // toggle P1.2 to get square wave
                SJMP REPEAT              // repeat above steps to generate square wave

```

Minimum frequency will be generated when minimum count (maximum delay) is loaded.

For mode 1

The minimum count is 0000, the delay that can be generated is 65536 cycles for half cycle.

Therefore, the time for one cycle is $65536 \times 2 \times 1.085 \mu s = 131072 \times 1.085 \mu s = 142213.12 \mu s$
 $= 7.0317 \text{ Hz}$

For mode 2

The 8051 Microcontroller based Embedded Systems, First Edition.
<http://www.mhhe.com/patel/mbes>

The minimum count is 00, the delay that can be generated is 256 cycles for half cycle.

Therefore, the time for one cycle is $256 \times 2 \times 1.085 \mu\text{s} = 512 \times 1.085 \mu\text{s} = 555.52 \mu\text{s}$

= 1800.11 Hz

Question 6

a. Explain the function of the pins of 9-pin RS-232 connector.

The nine signals of DB9 connector are shown in the following Table.

RS 232 signals for DB9 connector

Pin	Description	Description.
1	Data carrier detect ($\overline{\text{DCD}}$)	Data Carrier Detect is used by a modem to indicate that it has made a connection with other remote modem, (or has detected a carrier signal)
2	Received data (RXD)	This pin carries incoming data from the serial device (DCE) to the computer (DTE)
3	Transmitted data (TXD)	This pin carries outgoing data from the computer (DTE) to the serial device (DCE)
4	Data terminal ready ($\overline{\text{DTR}}$)	After the power is turned ON, DTE asserts the Data terminal ready ($\overline{\text{DTR}}$) signal to inform the DCE (modem) that it is ready
5	Signal ground (GND)	Signal ground reference
6	Data set ready ($\overline{\text{DSR}}$)	When modem is ready, it asserts the Data set ready ($\overline{\text{DSR}}$) signal to the DTE
7	Request to send ($\overline{\text{RTS}}$)	Once DSR signal has been received, the DTE makes request to use the data channel by asserting $\overline{\text{RTS}}$ signal to start transmission.
8	Clear to send ($\overline{\text{CTS}}$)	This pin is used by the serial device (DCE) to acknowledge the computer's (DTE) RTS Signal
9	Ring indicator (RI)	Incoming signal from a modem. It is used by modem (in auto answer mode) to indicate the receipt of a telephone ring signal

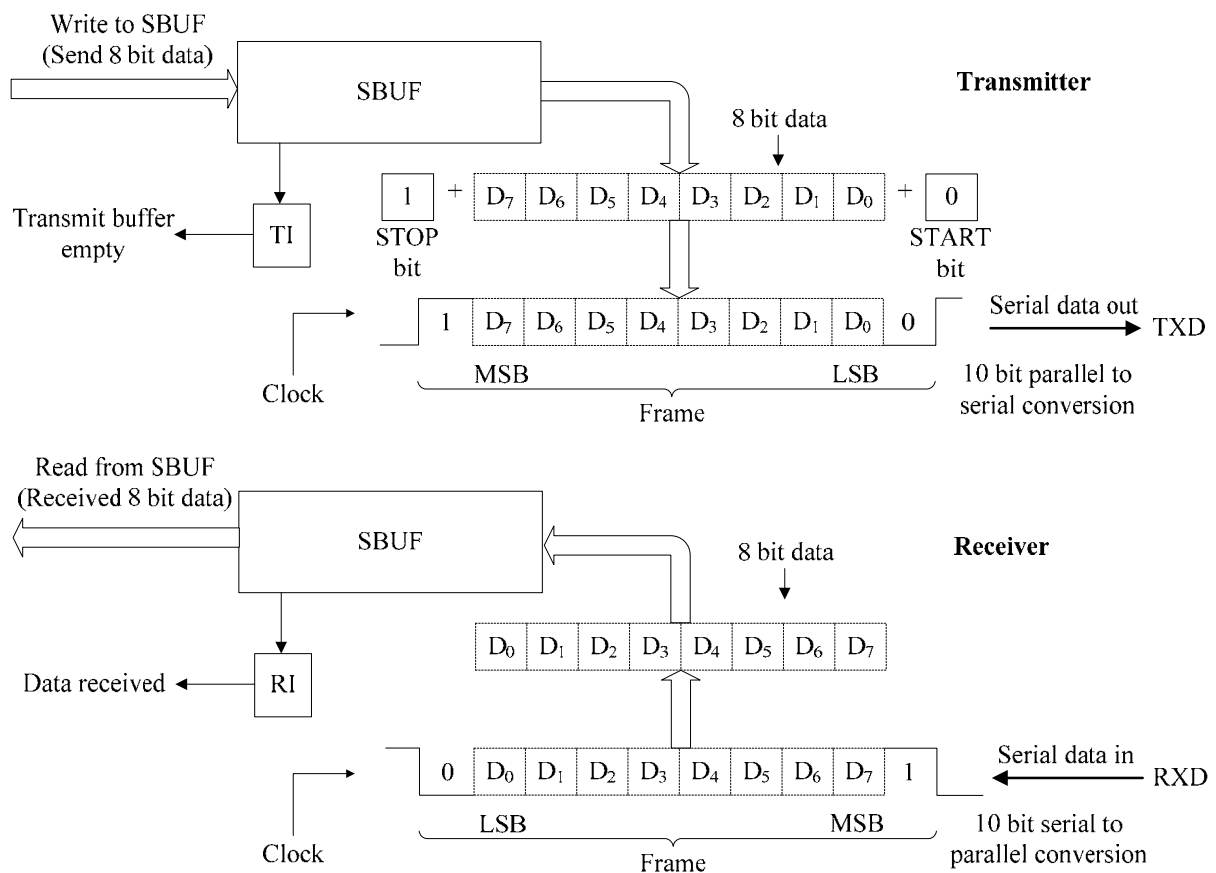
After the power is turned ON, DTE asserts the Data terminal ready ($\overline{\text{DTR}}$) signal to inform the DCE (modem) that it is ready, see figure 15.4. When modem is ready, it asserts the Data set ready ($\overline{\text{DSR}}$) signal to the DTE. Once $\overline{\text{DSR}}$ signal has been received, the DTE makes request to use the data channel by asserting $\overline{\text{RTS}}$ signal to start transmission. Then the modem at the other end (receiver end) is dialed. This modem (usually in answer mode) replies by sending a signal at

a carrier frequency of 2255 Hz. When the modem at the sending terminal receives this signal, it sends Data carrier detect ($\overline{\text{DCD}}$) to the DTE, then after the modem sends Clear to send ($\overline{\text{CTS}}$) showing that the channel is ready for transmission. Immediately after receiving $\overline{\text{CTS}}$ signal, the DTE of the transmitter side sends serial data on its TXD output. For reception of data similar handshaking process is carried out.

b. Explain how 8051 transmit the character and receives a character serially using UART.

Mode 1- Standard 8 bit UART mode

UART is designed for mainly for this mode and frame format of this mode is compatible with COM port of PCs. This mode transmits 10 bits through TXD pin and receives through RXD pin as follows: a START bit (always 0), 8 data bits (LSB first) and a STOP bit (always 1). Programmer can set its transmission/reception rate using Timer 1. The process of data transmission and reception is illustrated in figure 15.12.



UART mode 1 operation

Transmission: Data transmission begins by writing data to the SBUF register. The START and STOP bits are added by hardware to form a 10 bit frame, see figure 15.12, then, 10 bit parallel to serial conversion is performed and one bit (LSB first) at a time is transmitted through TXD pin, once complete frame is transmitted, TI flag is set automatically by serial port hardware to indicate end of the data transmission. We need to monitor TI flag to conform that SBUF register is not overloaded. If TI flag is set, it implies that last character transmission is completed and now SBUF is empty and new byte can be written to it to start next transmission. If a new byte is written to SBUF before TI is raised, the untransmitted part of previous byte will be lost.

It should be noted that microcontroller sets TI flag when it completes byte transfer, whereas it must be cleared by the programmer after next byte is loaded in to SBUF.

Reception: The data reception begins when REN=1 and high to low transition (start bit) is detected on RXD pin. The received byte is loaded in to SBUF register (the START and STOP bits are separated by UART hardware once complete frame is received) and stop bit in to RB8 (SCON bit 2) only if following two conditions are met.

- i. RI=0, showing that previous data byte is read by a program
- ii. Either SM2=0 or stop bit =1, Normally SM2 =0 and character will be accepted irrespective of status of stop bit. A program may check RB8 to ensure that stop bit is correct, if required.

If these two conditions are not met, received character is ignored and RI is not set and receiver circuit waits for next start bit.

c. Write 8051 C program to transmit serially the message ‘SWITCH ON’ or ‘SWITCH OFF’ depending on the status of the simple switch connected to pin P1.2. Use 2400 baud rate. 1 stop bit, 8 data bits format and assume XTAL frequency as 11.0592 MHz.

```
#include<reg51.h>
void main (void)
{
    unsigned char ch;
    TMOD = 0x20;           // timer 1 configured in mode 2
    TH1 = 0xF4;            // set 2400bps baud rate
    SCON = 0x50;           // 8-bit data, 1 stop bit, REN enabled
    TR1 = 1;
```

The 8051 Microcontroller based Embedded Systems, First Edition.
<http://www.mhhe.com/patel/mbes>

```

while(1)
{
    SBUF = 'S';
    while (TI==0);
    TI = 0;
    SBUF = 'W';
    while (TI==0);
    TI = 0;
    SBUF = 'T';
    while (TI==0);
    TI = 0;
    SBUF = 'T';
    while (TI==0);
    TI = 0;
    SBUF = 'C';
    while (TI==0);
    TI = 0;
    SBUF = 'H';
    while (TI==0);
    TI = 0;
    SBUF = ' ';
    while (TI==0);
    TI = 0;
    if (p1.2==0)
    {
        SBUF = 'O';
        while (TI==0);
        TI = 0;
        SBUF = 'F';
        while (TI==0);
        TI = 0;
        SBUF = 'F';
        while (TI==0);
    }
    else
    {
        SBUF = 'O';
        while (TI==0);

```



```

        TI = 0;
        SBUF = 'N';
        while (TI==0);
        TI = 0;
    }
}
}

```

Question 7

a. Interface an LCD display to 8051 write an ALP to display the message ‘VERY GOOD’.

//P2.0-P2.7 are connected to LCD data pins D0-D7

//P1.0 is connected to RS pin of LCD

//P1.1 is connected to R/W pin of LCD

//P1.2 is connected to E pin of LCD

ORG 0000H

LCALL WAIT // Initialization of LCD by software

LCALL WAIT // this part of program is not mandatory but

MOV A, #38H // recommended to use because it will

LCALL COMMAND // guarantee proper initialization even when

LCALL WAIT // power supply reset timings are not met

MOV A, #38H

LCALL COMMAND

LCALL WAIT

MOV A, #38H

LCALL COMMAND // initialization complete

MOV A, #38H //Initialize LCD, 8 bit interface, 5X7 //dots/character

LCALL COMMAND // send command to LCD

MOV A, #0FH // display on, cursor on with blinking

LCALL COMMAND // send command to LCD

MOV A, #06 // shift cursor right

LCALL COMMAND // send command to LCD

MOV A, #01H // clear LCD screen and memory

LCALL COMMAND // send command to LCD

MOV A, #80H // set cursor at line 1, first position

LCALL COMMAND // send command to LCD

MOV A, # 'V' // V to be displayed

The 8051 Microcontroller based Embedded Systems, First Edition.
<http://www.mhhe.com/patel/mbes>

```

        LCALL DISPLAY          // send data to LCD for display
        MOV A, #'E'           // E to be displayed
        LCALL DISPLAY          // send data to LCD for display
        MOV A, #'R'           // R to be displayed
        LCALL DISPLAY          // send data to LCD for display
        MOV A, #'Y'           // Y to be displayed
        LCALL DISPLAY          // send data to LCD for display
        MOV A, #' '           // ' ' to be displayed
        LCALL DISPLAY          // send data to LCD for display
        MOV A, #'G'           // G to be displayed
        LCALL DISPLAY          // send data to LCD for display
        MOV A, #'O'           // O to be displayed
        LCALL DISPLAY          // send data to LCD for display
        MOV A, #'O'           // O to be displayed
        LCALL DISPLAY          // send data to LCD for display
        MOV A, #'D'           // D to be displayed
        LCALL DISPLAY          // send data to LCD for display

HERE: SJMP HERE                // wait indefinitely
COMMAND:                        // command write subroutine
        MOV P2, A              // place command on P1
        CLR P1.0               // RS=0 for command
        CLR P1.1               // R/W=0 for write operation
        SETB P1.2              // E=1 for high pulse
        LCALL WAIT             // wait for some time
        CLR P1.2               // E=0 for H-to-L pulse
        LCALL WAIT // wait for LCD to complete given command
        RET

DISPLAY:                        // data write subroutine
        MOV P2, A              // send data to port 1
        SETB P1.0              // RS=1 for data
        CLR P1.1               // R/W=0 for write operation
        SETB P1.2              // E=1 for high pulse
        LCALL WAIT             // wait for some time
        CLR P1.2               // E=0 for H-to-L pulse
        LCALL WAIT             // wait for LCD to write given data
        RET

WAIT: MOV R6, #30H             // delay subroutine
THERE: MOV R5, #0FFH           //
HERE1: DJNZ R5, HERE1          //

```

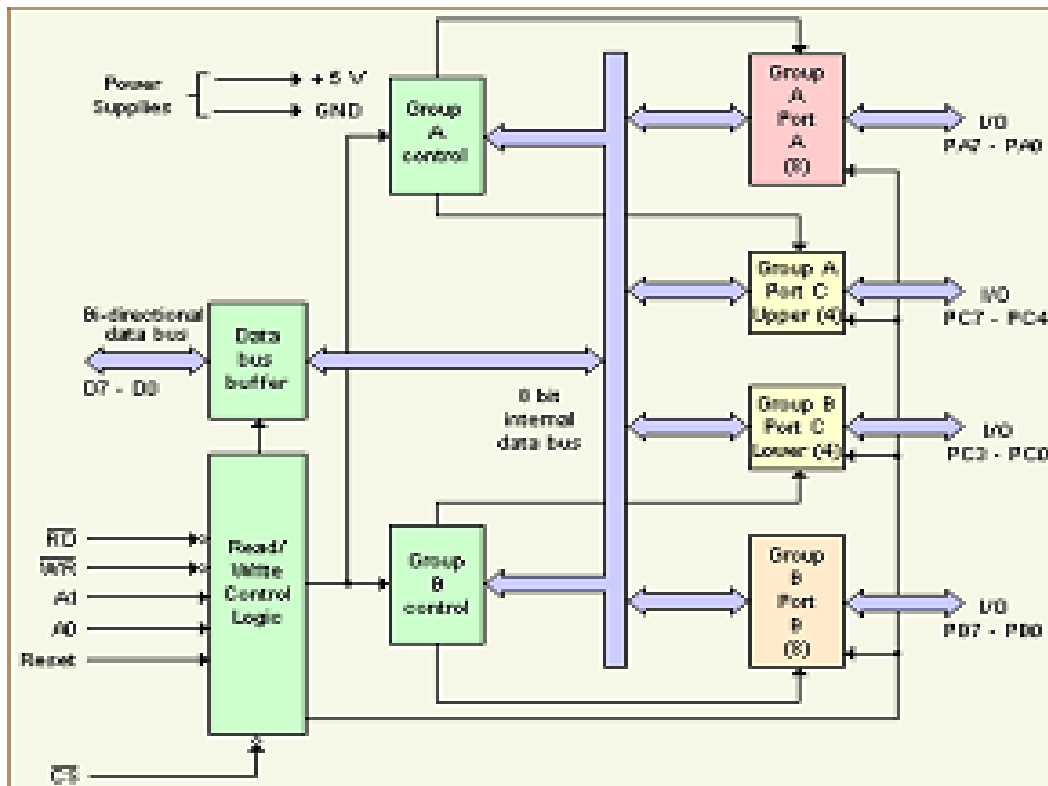
DJNZ R6, THERE
RET

b. With a block schematic explain the features 8255 PPI chip and its MODE-0 Operation.

8255 PPI (Programmable Peripheral Interface). 8255 is an I/O port chip used for interfacing I/O devices with microcontroller/processor.

The features of 8255 are:

- The 8255A is a general purpose programmable I/O device which is designed for use with all Intel and most other microcontrollers/processors.
- It gives 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation.



Mode 0 (Basic I/O mode): The model 0 is referred as called as basic input/output mode. The mode 0 gives input and output capabilities through the three ports. Data can be simply read from/written to the input/output ports.

The features of mode 0 are given below:

1. Two 8-bit ports (port A and port B) and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can also be collectively used as a 8-bit port.
2. Any port can be used as an input or output port.
3. Input ports are not latched while output ports are latched.
4. A maximum of four ports are available so that overall 16 I/O configuration are possible

c. If the internal memory 20H contains AAH and 07H contains 55H. What is the content of register A and status of carry bit after executing the following code:

```
MOV C,07H  
MOV A,#20H  
ADDC A,07H
```

A=76H

C=0 (RESET)

Question 8

- a. Explain the architecture of MSP430 CPU with its internal block diagram.**
- b. Explain briefly the i) Clock system and ii) Low power modes of operation of MSP430.**



FREE ONLINE LEARNING CENTRE FOR THIS BOOK

<http://www.mhhe.com/patel/mbes>

The book is supplemented with separate online resources for Instructors and Students.

Online resources for the Instructors:

- Complete solution manual of the book
- Chapter wise power point slides
- Additional material on Advanced microcontrollers

Online resources for the Students:

- **Lab manual** : A complete lab manual containing 14 laboratories with example references from the book, sample program followed by laboratory exercise to reinforce the concepts
- **Question Papers of different universities with the solution**
- **Additional question bank**
- **Chapter wise objectives, Key-terms and Points to remember**
- **Projects given in the book**: Each project includes problem statement, complete schematic diagram, program development, assembly and/or C program and suggested modifications.
- **Chapter wise Interfacing diagrams.**

Salient features:

- Logical sequencing of the topics, concise and to the point discussion.
- Pictorial explanation of majority of concepts followed by examples for ease of understanding
- Step by step approach for the software development
- Simulation methods and snapshots of the output for some key examples
- Programming examples in assembly and C language.
- Timing and data flow diagrams for instruction execution.
- Advanced and complex topics like interrupt handling, interrupt latency, lookup tables, timing analysis, stack operations, multiprocessor communications, 8051 enhancements and variants, internal port structure are covered with more clarity.
- All Instructions are explained through use of examples.
- Tutorial of Keil μ Vision4.0 Integrated development environment
- Excellent pedagogy
 - Learning objectives and key terms at the beginning of each chapter
 - Points to remember at the end of each chapter
 - Discussion questions within the topics: **25**
 - Review questions with answers: **310**
 - Exercise questions: **410**
 - Programming examples (Assembly and C): **325**
 - Objective questions: **301**
 - Think boxes with answers: **95**
 - Illustrations (Figures and Tables): **350**
- Coverage of many variants, peripheral devices, PIC and AVR microcontrollers
- Advance serial data transfer protocols: I2C and SPI

The 8051 Microcontroller based Embedded Systems, First Edition.

<http://www.mhhe.com/patel/mbes>